SPRAWOZDANIE

Zajęcia: Nauka o danych II

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium Nr 2 Data 24.05.2025 Temat: „Praktyczne zastosowanie rekurencyjnych sieci neuronowych (RNN) do analizy szeregów czasowych” Wariant 2	Anna Więzik Informatyka II stopień, niestacjonarne, 2 semestr, gr.1a TTO

1. Polecenie:

Link do repozytorium: https://github.com/AnaShiro/NoD2_2025

2. **Wariant 2**
   (a) Prognozowanie zużycia energii elektrycznej.
   (b) Wykrywanie anomalii w poziomie hałasu w środowisku.
   (c) Rozpoznawanie cyklu snu na podstawie danych z opaski fitness.

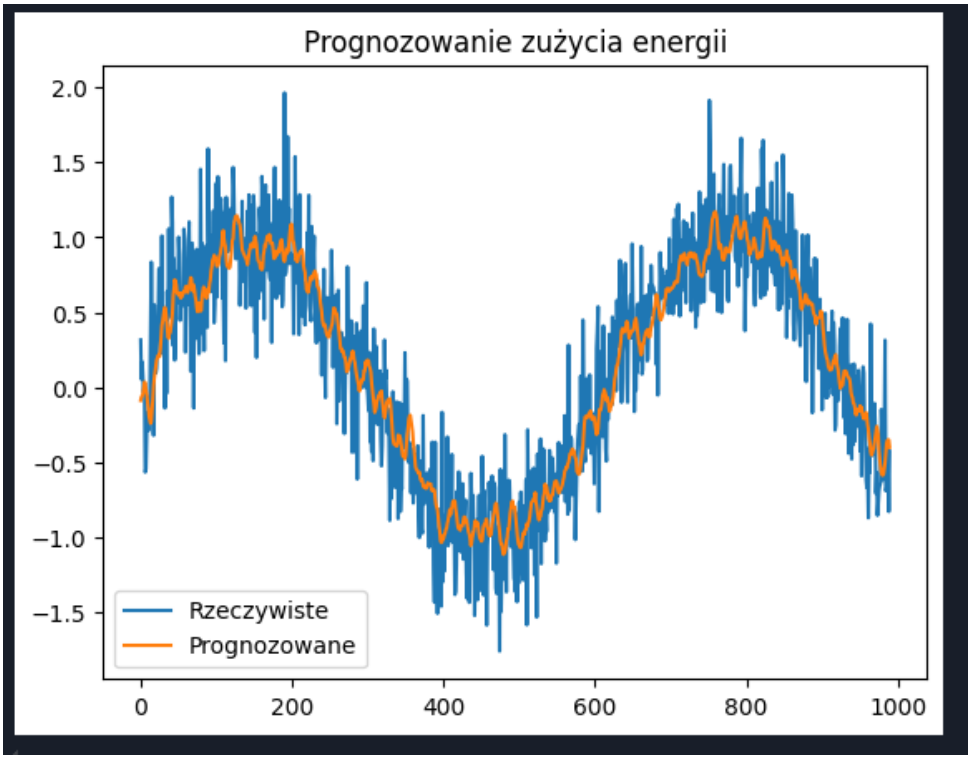
Rekurencyjne sieci neuronowe (RNN) są stworzone do analizy danych sekwencyjnych, co pozwala im efektywnie modelować zależności czasowe. Ich odmiany, takie jak LSTM i GRU, dobrze radzą sobie z dłuższymi sekwencjami, dzięki czemu znajdują zastosowanie w prognozowaniu, klasyfikacji oraz identyfikacji wzorców w szeregach czasowych. Choć RNN napotykają wyzwania, takie jak problem zanikającego gradientu i wysokie koszty obliczeniowe, pozostają kluczowym narzędziem w analizie danych tekstowych i czasowych.

2. Opis programu opracowanego

```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

def create_sequences(dataset, look_back=10):
    X, Y = [], []
    for i in range(len(dataset) - look_back):
        X.append(dataset[i:i+look_back, 0])
        Y.append(dataset[i+look_back, 0])
    return np.array(X), np.array(Y)

def generate_energy_data(n_samples=1000):
    x = np.linspace(0, 10, n_samples)
    y = np.sin(x) + np.random.normal(0, 0.3, size=n_samples)
    return y.reshape(-1, 1)

data_energy = generate_energy_data()
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data_energy)

X, y = create_sequences(data_scaled, look_back=10)
X = X.reshape(X.shape[0], X.shape[1], 1)

model_energy = tf.keras.Sequential([
    tf.keras.layers.LSTM(64, input_shape=(X.shape[1], 1)),
    tf.keras.layers.Dense(1)
])

model_energy.compile(optimizer='adam', loss='mse')
model_energy.fit(X, y, epochs=20, batch_size=32)

predicted = model_energy.predict(X)
predicted = scaler.inverse_transform(predicted)

plt.plot(scaler.inverse_transform(y.reshape(-1, 1)), label="Rzeczywiste")
plt.plot(predicted, label="Prognozowane")
plt.legend()
plt.title("Prognozowanie zużycia energii")
plt.show()
```

✓ 2.0s

```
super().__init__(**kwargs)
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2403
Epoch 2/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0107
Epoch 3/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0086
Epoch 4/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0083
Epoch 5/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0076
Epoch 6/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0077
Epoch 7/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0074
Epoch 8/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0080
Epoch 9/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0078
Epoch 10/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0076
Epoch 11/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0075
Epoch 12/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0077
Epoch 13/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0078
...
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0071
Epoch 20/20
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0069
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

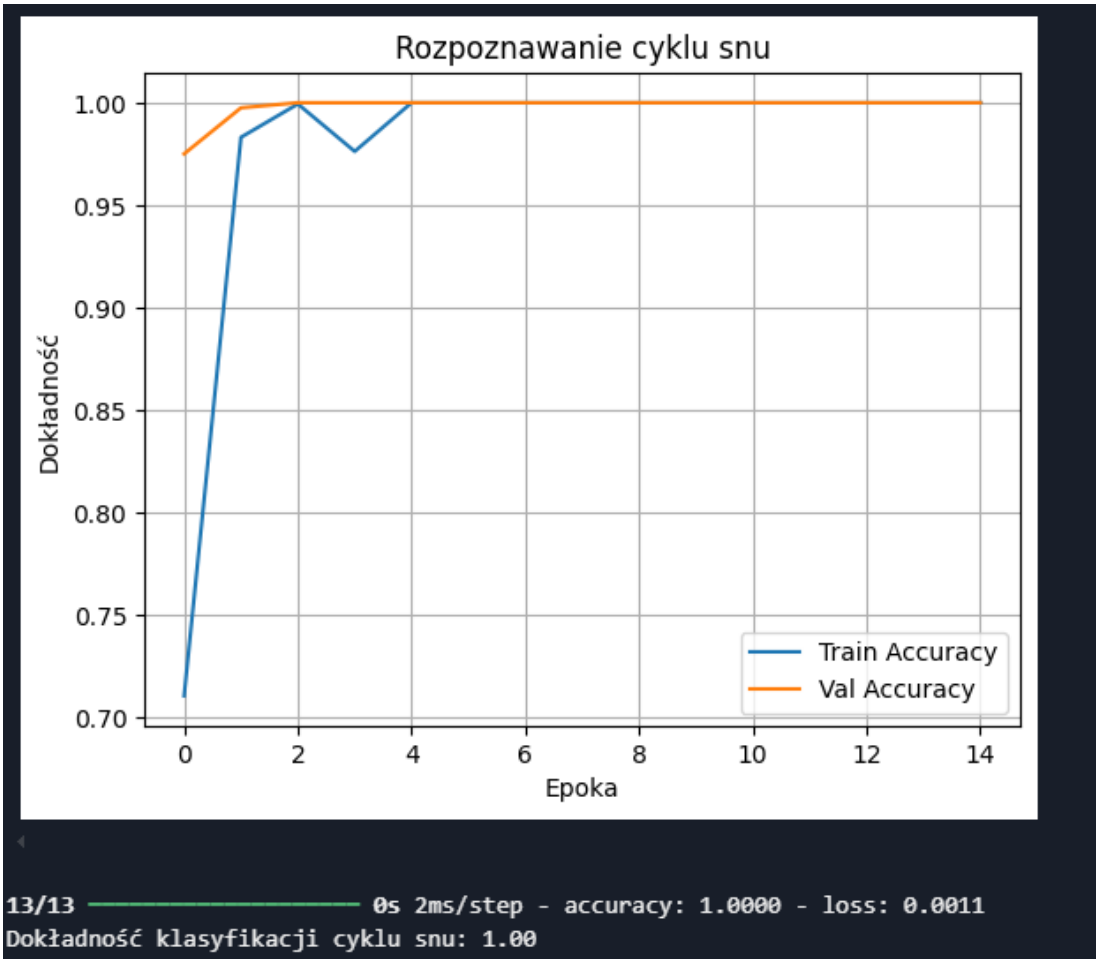


```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

def generate_sensor_data(n_samples=1000, timesteps=10):
    X = np.random.normal(0, 1, (n_samples, timesteps))
    y = np.zeros(n_samples)

    anomaly_indices = np.random.choice(n_samples, size=n_samples // 10, replace=False)
    X[anomaly_indices] += np.random.normal(5, 1, (len(anomaly_indices), timesteps))
    y[anomaly_indices] = 1

    return X.reshape((n_samples, timesteps, 1)), y

X_noise, y_noise = generate_sensor_data()
X_train, X_test, y_train, y_test = train_test_split(X_noise, y_noise, test_size=0.2)

model_noise = tf.keras.Sequential([
    tf.keras.layers.LSTM(32, input_shape=(X_noise.shape[1], 1)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_noise.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_noise.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

loss, accuracy = model_noise.evaluate(X_test, y_test)
print(f"Dokładność wykrywania anomalii w hałasie: {accuracy:.2f}")
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8334 - loss: 0.6676 - val_accuracy: 0.9700 - val_loss: 0.5320
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9870 - loss: 0.4749 - val_accuracy: 1.0000 - val_loss: 0.1547
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0821 - val_accuracy: 1.0000 - val_loss: 0.0103
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0077 - val_accuracy: 1.0000 - val_loss: 0.0058
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0049 - val_accuracy: 1.0000 - val_loss: 0.0042
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0036 - val_accuracy: 1.0000 - val_loss: 0.0033
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0027 - val_accuracy: 1.0000 - val_loss: 0.0028
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 1.0000 - val_loss: 0.0023
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 1.0000 - val_loss: 0.0020
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 1.0000 - val_loss: 0.0017
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0016
Dokładność wykrywania anomalii w hałasie: 1.00
```

```python
def generate_sleep_data(n_samples=2000, timesteps=30):
    X, y = [], []

    for label in range(3):  # Fazy snu: 0 = lekki sen, 1 = głęboki sen, 2 = REM
        for _ in range(n_samples // 3):
            if label == 0:
                sequence = np.random.normal(0, 0.2, (timesteps,))
            elif label == 1:
                sequence = np.sin(np.linspace(0, 2*np.pi, timesteps)) + np.random.normal(0, 0.1, (timesteps,))
            else:
                sequence = np.sin(np.linspace(0, 4*np.pi, timesteps)) + np.random.normal(0, 0.15, (timesteps,))
            X.append(sequence)
            y.append(label)

    X = np.array(X).reshape(-1, timesteps, 1)
    y = tf.keras.utils.to_categorical(np.array(y), num_classes=3)

    return X, y

X_sleep, y_sleep = generate_sleep_data()
X_train, X_test, y_train, y_test = train_test_split(X_sleep, y_sleep, test_size=0.2)

model_sleep = tf.keras.Sequential([
    tf.keras.layers.LSTM(64, input_shape=(X_sleep.shape[1], 1)),
    tf.keras.layers.Dense(3, activation='softmax')
])

model_sleep.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model_sleep.fit(X_train, y_train, epochs=15, batch_size=32, validation_data=(X_test, y_test))

plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epoka')
plt.ylabel('Dokładność')
plt.title('Rozpoznawanie cyklu snu')
plt.legend()
plt.grid()
plt.show()

loss, accuracy = model_sleep.evaluate(X_test, y_test)
print(f"Dokładność klasyfikacji cyklu snu: {accuracy:.2f}")
```

```
Epoch 1/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5190 - loss: 0.9240 - val_accuracy: 0.9750 - val_loss: 0.2142
Epoch 2/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9696 - loss: 0.1994 - val_accuracy: 0.9975 - val_loss: 0.0475
Epoch 3/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9997 - loss: 0.0161 - val_accuracy: 1.0000 - val_loss: 0.0080
Epoch 4/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9778 - loss: 0.0665 - val_accuracy: 1.0000 - val_loss: 0.0417
Epoch 5/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0283 - val_accuracy: 1.0000 - val_loss: 0.0100
Epoch 6/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0089 - val_accuracy: 1.0000 - val_loss: 0.0060
Epoch 7/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0055 - val_accuracy: 1.0000 - val_loss: 0.0042
Epoch 8/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0041 - val_accuracy: 1.0000 - val_loss: 0.0032
Epoch 9/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0030 - val_accuracy: 1.0000 - val_loss: 0.0026
Epoch 10/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0026 - val_accuracy: 1.0000 - val_loss: 0.0021
Epoch 11/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 1.0000 - val_loss: 0.0018
Epoch 12/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 1.0000 - val_loss: 0.0015
Epoch 13/15
...
Epoch 14/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 1.0000 - val_loss: 0.0011
Epoch 15/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 1.0000 - val_loss: 0.0010
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```



3. Wnioski

Ćwiczenie pozwoliło uczestnikom na praktyczne zapoznanie się z implementacją i zastosowaniem RNN w analizie danych czasowych. Na przykładach prognozowania, wykrywania anomalii oraz rozpoznawania aktywności mogli zaobserwować, w jaki sposób RNN uczą się zależności sekwencyjnych. Zrozumienie mechanizmów działania LSTM i GRU oraz ich praktyczne wykorzystanie stanowi solidną podstawę do dalszej pracy z danymi sekwencyjnymi i czasowymi.